How can I improve my app? Classifying user reviews for software maintenance and evolution

Panichella, Sebastiano ; Di Sorbo, Andrea ; Guzman, Emitza ; Visaggio, Corrado Aaron ; Canfora, Gerardo ; Gall, Harald C

Abstract: App Stores, such as Google Play or the Apple Store, allow users to provide feedback on apps by posting review comments and giving star ratings. These platforms constitute a useful electronic mean in which application developers and users can productively exchange information about apps. Previous research showed that users feedback contains usage scenarios, bug reports and feature requests, that can help app developers to accomplish software maintenance and evolution tasks. However, in the case of the most popular apps, the large amount of received feedback, its unstructured nature and varying quality can make the identification of useful user feedback a very challenging task. In this paper we present a taxonomy to classify app reviews into categories relevant to software maintenance and evolution, as well as an approach that merges three techniques: (1) Natural Language Processing, (2) Text Analysis and (3) Sentiment Analysis to automatically classify app reviews into the proposed categories. We show that the combined use of these techniques allows to achieve better results (a precision of 75% and a recall of 74%) than results obtained using each technique individually (precision of 70% and a recall of 67%).

DOI: https://doi.org/10.1109/ICSM.2015.7332474

How Can I Improve My App? Classifying User Reviews for Software Maintenance and Evolution

S. Panichella*, A. Di Sorbo†, E. Guzman‡, C. A.Visaggio†, G. Canfora† and H. C. Gall*
*University of Zurich, Switzerland
†University of Sannio, Benevento, Italy
‡Technische Universität München, Garching, Germany
panichella@ifi.uzh.ch, disorbo@unisannio.it, emitza.guzman@mytum.de, {visaggio,canfora}@unisannio.it, gall@ifi.uzh.ch

***Abstract*—App Stores, such as Google Play or the Apple Store, allow users to provide feedback on apps by posting review comments and giving star ratings. These platforms constitute a useful electronic mean in which application developers and users can productively exchange information about apps. Previous research showed that users feedback contains usage scenarios, bug reports and feature requests, that can help app developers to accomplish software maintenance and evolution tasks. However, in the case of the most popular apps, the large amount of received feedback, its unstructured nature and varying quality can make the identification of useful user feedback a very challenging task. In this paper we present a taxonomy to classify app reviews into categories relevant to software maintenance and evolution, as well as an approach that merges three techniques: (1) Natural Language Processing, (2) Text Analysis and (3) Sentiment Analysis to automatically classify app reviews into the proposed categories. We show that the combined use of these techniques allows to achieve better results (a precision of 75% and a recall of 74%) than results obtained using each technique individually (precision of 70% and a recall of 67%).**

***Index Terms*—User Reviews, Mobile Applications, Natural Language Processing, Sentiment Analysis, Text classification**

I. Introduction

App stores are digital distribution platforms that allow users to download and rate mobile apps. Notable distribution platforms for mobile devices include Apple and Android app stores, in which users can comment and write *reviews* of the mobile apps they are using. These reviews serve as a communication channel between developers and users where users can provide relevant information to guide app developers in accomplishing several software maintenance and evolution tasks, such as the implementation of new features, bug fixing, or the improvement of existing features or functionalities. App developers spend considerable effort in collecting and exploiting user feedback to improve user satisfaction. Previous work [10], [18], [31] has shown that approximately one third of the information contained in user reviews is helpful for developers. However, processing, analyzing and selecting useful user feedback presents several challenges. First of all, app stores include a *substantial body* of reviews, which requires a large amount of effort to manually analyze and process. An empirical study by Pagano *et al.* [31] found that mobile apps received approximately 23 reviews per day and that popular apps, such as Facebook, received on average 4,275 reviews per day. Additionally, users usually provide their feedback in form of *unstructured* text that is difficult to parse and analyze. Thus, developers and analysts have to read a large amount of textual data to become aware of the comments and needs of their users [10]. In addition, the *quality* of reviews varies greatly, from useful reviews providing ideas for improvement or describing specific issues to generic praises and complaints (e.g. *"You have to be stupid to program this app"*, *"I love it!"*, *"this app is useless"*).

To handle this problem Chen *et al.* [10] proposed AR-Miner, an approach to help app developers discover the most *informative* user reviews. Specifically, the authors use: (i) text analysis and machine learning to filter out non-informative reviews and (ii) topic analysis to recognize topics treated in the reviews classified as *informative*. In this paper, we argue that text content represents just one of the possible dimensions that can be explored to detect informative reviews from a software maintenance and evolution perspective. In particular, topic analysis techniques are useful to discover topics treated in the review texts, but they are not able to reveal the authors' *intentions* (i.e. the writers' goals) for reviews containing specific topics. For example, let's consider the following two user reviews sentences:

1) *"The awful button in the page doesn't work"*
2) *"A button in the page should be added"*

Topic analysis will reveal that these two reviews are likely to discuss the same topics: *"button"* and *"page"*. However, these reviews have different *intentions*: in review *(1)* the user has exposed a problem related to the app, while in the review *(2)* the author asks for the implementation of a new feature. This example illustrates that understanding the *intention* in user reviews could add useful information for accomplishing software maintenance and evolution tasks. We conjecture that a deep analysis of the sentences structure in user reviews can be exploited to determine the *intention* of a given review. In addition, also the *sentiment* expressed in the two user reviews can be exploited to distinguish different kinds of informative reviews. For example, in review sentence (1) which reports a bug, the sentiment expressed by the user is negative (i.e., *awful button*) while, for review (2) the sentiment expressed is more *neutral*. In this paper we investigate whether the (i) *structure*, (ii) *sentiment* and (iii) *text* features contained in user reviews can be used to classify and select the user reviews that are helpful for developers to maintain and evolve their app. Thus,

we propose an approach that combines Natural Language Processing (NLP), Sentiment Analysis (SA) and Text Analysis (TA) techniques for the extraction of information present in user reviews that is relevant to the maintenance and evolution of mobile apps. Furthermore, we use machine learning (ML) to combine the three techniques and through a quantitative evaluation show that the combination of the three techniques outperforms the performance of each individual technique. To the best of our knowledge, this is the first work that merges Natural Language Processing, Sentiment Analysis and Text Analysis to extract app store reviews that are relevant for software maintenance and evolution. The main contributions of this paper are as follows:

- *A high level taxonomy of categories of sentences contained in app user reviews that are relevant for the maintenance and evolution of mobile apps.*
- *A novel approach to extract users' intentions expressed in app store reviews based on Natural Language Processing.*
- *An empirical study that investigates to what extent NLP, SA and TA features help to detect app store reviews relevant for the maintenance and evolution of mobile apps.*

Paper structure: Section II presents the approach and techniques we used. Section III reports the dataset and the evaluation methods we employed. Section IV presents and discusses the results of the study. Section V deals with the threats that could affect the validity of our work. Section VI illustrates the related literature and Section VII concludes the paper outlining future research directions.

II. Approach

The main goal of our research is to help developers of mobile apps to categorize information from user reviews that is relevant for software maintenance and evolution. Thus, the research questions that guided our work are:

- **RQ1:** *Are the language structure, content and sentiment information able to identify user reviews that could help developers in accomplishing software maintenance and evolution tasks?*
- **RQ2:** *Does the combination of language structure, content and sentiment information produce better results than individual techniques used in isolation?*

This Section describes the research approach we performed to answer our research questions.

A. Approach Overview

Figure 1 depicts the research approach we followed to answer our research questions. Specifically, our approach consisted of four steps:

1) ***Taxonomy for Software Maintenance and Evolution:*** we manually analyzed users reviews of seven Apple Store and Google Play apps and rigorously deduced a taxonomy of the reviews containing useful content for software maintenance and evolution. The output of this phase consisted of a taxonomy of user reviews categories that can lead the developers to select the reviews more useful for a specific maintenance task (i.e. bug fixing, feature adding, etc.).
2) ***Feature Extraction:*** the goal of this step was to extract a set of meaningful features from user reviews data to train ML techniques and automatically label app review content according to the taxonomy deduced in the first step. Thus, we designed three different techniques based on (i) ***Text Analysis***, (ii) ***Natural Language Processing*** and (iii) ***Sentiment Analysis***, that analyzed the content of app reviews and extracted features for the learning phase of our approach. The output of this phase was represented by a set of NLP, TA and SA features.
3) ***Learning Classifiers:*** in this step we used the NLP, TA and SA features extracted in the previous phase of the approach to train ML techniques and classified app reviews according to the taxonomy deduced in the first step. Moreover, we also experimented with different combinations of NLP, TA and SA features to train ML approaches.
4) ***Evaluation:*** in this step we evaluated the performance of the ML techniques experimented in the previous step relying on widely adopted metrics for machine learning evaluation.

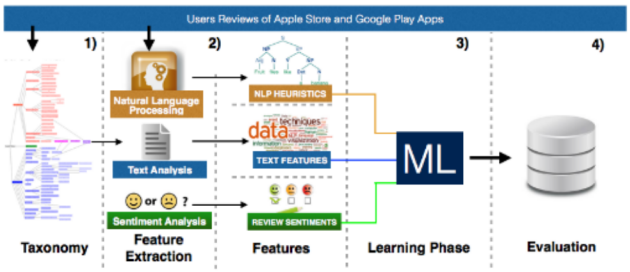

Fig. 1. Overview Research Approach

B. Taxonomy for Software Maintenance and Evolution

The goal of this first step is to deduce a taxonomy of user reviews categories that is relevant to software maintenance and evolution. To achieve this objective, we analyse user reviews data at the **sentence-level granularity** because within a raw user review some sentences are relevant to software evolution and maintenance, while others are not. We argue that the definition of such a taxonomy requires the understanding of which kind of feedback developers look for in user reviews. Developers usually exchange messages on development communication channels, such as mailing lists and issue trackers, to plan and discuss maintenance and evolution tasks. Therefore, we conjecture that the analysis of discussions occurring in such communication means can guide us in defining a taxonomy of sentence categories that developers perceive as important for software maintenance and evolution. For this reason we (i) investigate the types of discussions occurring among developers through the manual inspection of messages exchanged by developers in development mailing lists of two

open source projects namely, Qt Project[1] and Ubuntu[2] (also in this case we perform the manual analysis at the *sentence-level granularity*); (ii) we perform a systematic mapping between categories of sentences reported in mailing lists messages with a previously defined taxonomy of content generally present in user reviews. A taxonomy of high-level categories of

TABLE I
INITIAL SET OF CATEGORIES OF SENTENCES

Category of Sentences	Description
Feature Request	Sentences related to ideas/suggestions/needs for improving or enhancing the product/service or its functionalities (e.g., *"we should add a button"*)
Opinion Asking	Sentences used for requiring someone to explicitly express her/his point of view about something (e.g. *"what do you think about the main panel?"*)
Problem Discovery	Sentences related to issue definitions and unexpected behaviors (e.g., *"the problem occurs when I try to access to database"*)
Solution Proposal	Sentences used to describe possible solutions for known problems (e.g., *"let's try to use a new method for login")*
Information Seeking	Sentences related to attempts to obtain information or help from other developers (e.g., *"can you explain how code works?"*)
Information Giving	Sentences used to inform/update other developers about something (e.g., *"the plan is to release new updates this week"*)

sentences was obtained by manually classifying development emails, using grounded theory [19]. To address this purpose, 300 emails (exchanged in the period between 01-11-2014 and 01-01-2015) have been extracted from the development mailing lists of Qt and Ubuntu (150 for each of them). Two authors of this work and an external validator (a PhD student in Computer Science) manually grouped all extracted emails according to the categories defined by Guzzi et al. [21] for developer communication: *implementation, technical infrastructure, project status, social interactions, usage* and *discarded*. In a second step, for each group of emails (with the exception of *discarded*) significant sentences have been selected and extracted relying on a finer-grained taxonomy proposed by Guzzi et al. [22]. This second taxonomy tries to model the reasons why developers need to communicate about source code and consists of three categories: *coordination, seeking information*, and *courtesy*. A manual analysis of the extracted sentences convinced the annotators of the importance of reshaping and extending this second taxonomy through the identification of categories with a closer connection to software maintenance and evolution activities. Table I shows the identified categories and their respective descriptions.

We performed a systematic mapping (see Table II) between this initial set of categories and the taxonomy proposed by Pagano *et al.* [31] which describes a set of 17 common topics present in app reviews. Additionally, we evaluated the relevance of each of the topics proposed by Pagano *et al.* for developers performing software evolution and maintenance tasks. We noticed that some of the categories we previously identified (see Table I) were irrelevant in the domain of app user reviews (see Table II). The results of the systematic mapping highlight that eight of the topics reported in the taxonomy of Pagano *et al.* [31] were relevant for developers. Table II shows the (i) categories of topics proposed by Pagano et al. , (ii) their relevance for software maintenance and evolution tasks and (iii) the mapping to the sentence categories in the initial taxonomy presented in Table I. These topics

TABLE II
TOPICS MAPPING WITH IDENTIFIED CATEGORIES OF SENTENCES

Topic	Relevant for Developers	Software Evolution Category
praise	no	-
helpfulness	no	-
feature information	yes	information giving
shortcoming	yes	information giving
bug report	yes	problem discovery
feature request	yes	feature request
other app	no	-
recommendation	no	-
noise	no	-
dissuasion	no	-
content request	yes	feature request
promise	yes	feature request
question	yes	information seeking
improvement request	yes	feature request
depraise	no	-
other feedback	no	-
how to	no	-

match with four of the six categories of sentences we identified in the context of development mailing lists:

- **Information Giving:** sentences that inform or update users or developers about an aspect related to the app.
- **Information Seeking:** sentences related to attempts to obtain information or help from other users or developers.
- **Feature Request:** sentences expressing ideas, suggestions or needs for improving or enhancing the app or its functionalities.
- **Problem Discovery:** sentences describing issues with the app or unexpected behaviours.

We consider such categories as the base categories in our taxonomy and thus, they represent the output of this first phase of our research approach.

C. Text Analysis

This section discusses the approach we used to extract textual features from app reviews. Specifically, it consists of two steps:

1) ***Preprocessing***: all terms contained in our set of user reviews are used as an information base to build a textual corpus that is preprocessed applying stop-word removal (using the english standard stop-word list) and stemming (English Snowball Stemmer) to reduce the number of text features for the ML techniques. The output of this phase corresponds to a Term-by-Document matrix *M* where each column represents a sentence and each row

[1]http://qt-project.org
[2]http://www.ubuntu.com

represents a term contained in the given sentence. Thus, each entry $\mathbf{M}_{[i,j]}$ of the matrix represents the weight (or importance) of the i$-th$ term contained in the j$-th$ sentence.

2) ***Textual Feature Weighting***: words are weighted using the the *tf* (term frequency), which weights each words *i* in a review *j* as:

$$tf_{i,j} = \frac{rf_{i,j}}{\sum_{k=1}^{m} rf_{k,j}}$$

where $\mathbf{rf}_{i,j}$ is the raw frequency (number of occurrences) of word *i* in review *j*.

We used the *tf* (term frequency) instead of *tf-idf* indexing because the use of the inverse document frequency (idf) penalises too much terms appearing in many reviews [15]. In our work, we are not interested in penalising such terms (e.g., "fix","problem", or "feature") that actually appear in many reviews because they may constitute interesting features that guide ML techniques in classifying sentences containing useful feedback from the software maintenance and evolution perspective. The weighted matrix *M* represents the output of this phase and the input for ML strategies as described in the Section II-F.

D. Natural Language Processing

We assume that when users write app reviews (e.g., to report bugs or propose new features) they tend to use recurrent linguistic patterns. For instance let's consider the sentence *"You should add a new button"*. A developer who reads this sentence can easily understand that the writer's intention is to make a *feature request*. Observing the sentence syntax, we can notice that the sentence presents a well defined predicate-argument structure:

- "add" constitutes the principal predicate of the sentence
- "you" represents the subject of the sentence
- "button" represents the direct object of the predicate
- "new" represents an attribute of the direct object
- "should" is the auxiliary of the principal predicate

We argue that this sentence matches a recurrent linguistic pattern that can be exploited for the recognition of sentences belonging to the *feature request* category of the taxonomy presented in Section II-B. Our conjecture is that this and similar patterns are intrinsically related to the *intentions* of the users that wrote the text. Furthermore, we believe that user intentions relevant for our purposes can be mapped to the categories defined in our taxonomy. Therefore, recurrent linguistic patterns can be exploited to recognize sentences of others categories belonging to our taxonomy.

Through a manual inspection of 500 reviews (different from the reviews described in Section III)) from different kinds of apps (games, communication, productivity, photography, etc.) we identified 246 recurrent linguistic patterns[3]. For each identified linguistic pattern we formalized and implemented an NLP heuristic to automatically recognize it. For instance, for the previous example we define the general NLP heuristic *"[someone] should add [something]"*. The implementation of a NLP heuristic enables the automatic detection of a sentence which matches a specific structure (e.g. "add" or a synonym as principal predicate, "should" in the auxiliary role of the principal predicate, a generic subject indicating who makes the request and a generic direct object indicating the request object).

To automatically detect sentences containing our defined NLP heuristics we used the Stanford Typed Dependencies (STD) parser [13] which is a tool able to represent dependencies between individual words contained in sentences and to label each of them with a specific grammatical relation. It uses the Stanford Dependencies (SD) representation, which was successfully used in a range of downstream tasks, including Textual Entailments [12] and BioNLP [17], thus, becoming a de-facto standard for parser evaluation in English [7] [30]. In this step, the NLP parser we implemented assigns each sentence in the input to its corresponding NLP heuristic. If the sentence structure does not match any of the defined NLP heuristics the NLP parser simply labels the sentence as *others*. The output of this step is a mapping between each sentence contained in a review and its corresponding NLP heuristic. We then use the NLP heuristic extracted for each sentence to train different ML techniques, as will be explained in the Section II-F.

E. Sentiment Analysis

Sentiment analysis is the process of assigning a quantitative value to a piece of text expressing an affect or mood [27]. We consider sentiment analysis as a text classification task which assigns each given sentence in a user review to one corresponding class. For our purpose, the classes are defined as three different levels of sentiment intensity: positive, negative and neutral. In our approach we use Naive Bayes for predicting the sentiment in the user reviews. Previous work [33] found that Naive Bayes performed better than other machine learning algorithms traditionally used for text classification when analyzing the sentiment in movie reviews. For our sentiment analysis task we performed the same preprocessing steps performed in the TA technique (stop word removal, stemming and transformation to a Term-by-Document matrix). Additionally, we performed a selection of the words considered to be most important for determining sentiment according to the Chi-squared x^2 metric. We trained our classifier with a set of 2090 App Store and Google Play review sentences which were randomly selected from the dataset described in Section III-A. The sentences were manually labeled by two annotators, an author of the paper and a graduate student in Computer Science with experience in sentiment analysis. To assure that both annotators had a similar understanding of the task to be done, a short clarification session was held and examples of annotated sentences were shown. The disagreement rate between both annotators was 5%. We computed the final sentiment score of each manually labeled sentence by averaging the two scores. We performed the sentiment analysis task using the Weka

[3]http://www.ifi.uzh.ch/seal/people/panichella/Appendix.pdf

tool [37] generating as output of this step an integer value in the [1,-1] range to each of the input sentences. The value of 1 determines positive sentiments, whereas 0 and -1 denote neutral and negative sentiments respectively.

F. Learning Classifiers

This section discusses how we trained machine learning techniques to classify user reviews, while Section III describes the data used as training and test set (below we refer them as $\mathbf{T}_1$ and $\mathbf{T}_2$), as well as, the procedure we followed to manually create the truth set. Formally, given a training set of app reviews sentences $\mathbf{T}_1$ and a test set of app reviews sentences $\mathbf{T}_2$, we automatically classify the reviews content in $\mathbf{T}_2$, by performing the following steps:

1) *NLP, TA and SA features:* The first step uses the NLP, TA and SA approaches discussed in the previous sections to compute the corresponding features contained in the sets of app reviews sentences $\mathbf{T}_1$ and $\mathbf{T}_2$. In particular, the output of this phase corresponds to a matrix *M* where each column represents an app review sentence and each row represents a feature extracted using NLP, TA and SA approaches. Thus, each entry $\mathbf{M}_{[i,j]}$ in the matrix represents the value of the metric i$-th$ of the corresponding j$-th$ app review sentence.
2) *Split training and test features:* The second step splits the matrix $\mathbf{M}$ (the output of the previous step) in two sub-matrices $\mathbf{M}_{training}$ and $\mathbf{M}_{test}$. Specifically, $\mathbf{M}_{training}$ and $\mathbf{M}_{test}$ represent the matrix that contains the sentences (i.e., the corresponding columns in $\mathbf{M}$) of $\mathbf{T}_1$ and the matrix that contains the sentences (i.e., the corresponding columns in $\mathbf{M}$) of $\mathbf{T}_2$ respectively.
3) *Oracle building:* The third step aims at building the oracle to allow ML techniques to train from $\mathbf{M}_{training}$ and predict on $\mathbf{M}_{test}$. Thus, in this stage, the sentences in $\mathbf{T}_1$ and $\mathbf{T}_2$ are manually classified and assigned to one of the categories defined in Section II-B (as described in Section III two human evaluators performed this manual labelling).
4) *Classification:* The fourth step automatically classifies sentences relying on the output data obtained from the previous step, that is $\mathbf{M}_{training}$ and $\mathbf{M}_{test}$ (with classified sentences). Specifically, we experimented (relying on the Weka tool) different machine learning techniques, namely, the standard probabilistic naive Bayes classifier, Logistic Regression, Support Vector Machines, J48, and the alternating decision tree (ADTree). The choice of these techniques is not random since they were successfully used for bug reports classification [1], [38] and for defect prediction in many previous works [3], [5], [8], [29], [39], thus allowing to increase the generalisability of our findings.

To answer RQ1 we experimented the ML techniques described above performing a training on the NLP, TA, and SA features. Furthermore, to answer RQ2 we investigate whether specific combinations of NLP, TA and SA features allow to obtain a better classification. Specifically, we learn the ML techniques using different combination of features: (i) NLP+TA, (ii) NLP+SA and (iii) NLP+TA+SA.

III. Evaluation

In this section we describe the dataset and methodology we used during the evaluation.

A. Dataset

To answer our research questions we evaluated our approach on the set of reviews collected by Guzman and Maalej [20] wich contains reviews of the AngryBirds, Dropbox and Evernote apps available in Apple's App Store[4] and reviews from the apps TripAdvisor, PicsArt, Pinterest and Whatsapp available in Android's Google Play[5] store. All seven apps were in the list of the *most popular apps* in the year 2013 in their respective app store and belong to different app categories. The diversity of the chosen apps allows for evaluating the robustness of the approach by classifying reviews which contain different vocabularies and are written by different user audiences. Table III shows for each app considered in our dataset: (i) the application name, (ii) the app category it belongs to, (iii) the platform from which comments were collected, and (iv) the number of collected reviews.

TABLE III
Overview of the dataset

App	Category	Platform	Total Reviews
AngryBirds	Games	App Store	1538
Dropbox	Productivity	AppStore	2009
Evernote	Productivity	App Store	8878
TripAdvisor	Travel	App Store	3165
PicsArt	Photography	Google Play	4438
Pinterest	Social	Google Play	4486
Whatsapp	Communication	Google Play	7696

B. Evaluation Methodology

To address the two research questions presented in Section II we applied our research approach on the dataset discussed in Section III-A. We then compared our results against a manually labelled truth set by using metrics commonly used in machine learning and NLP tasks. In the following sections we describe the procedure for creating the truth set and the used metrics.

1) Creation of Truth Set: To create our truth set we first use AR-miner [10] to filter out non-informative reviews in our dataset. Then, we manually labeled a sample of dataset sentences. The sentences were selected through a stratified random sampling strategy. During the sampling we verified that the percentage of the number of extracted sentences per app was the same as the percentage of reviews per app in the original set. In total we sampled 1421 sentences out of 7696 reviews (18.46%).

Two authors of this work manually labeled the sample according to the categories of our taxonomy (see Section II-B).

[4]https://itunes.apple.com/us/genre/ios/id36
[5]https://play.google.com/store?hl=en

An additional category, named *other*, was used whenever the sentences did not match any of the predetermined categories. To assure that both annotators applied the same criteria when labeling the results, the definitions of each category were discussed among them before any labeling was done. Then, each annotator labeled a set of 20 sentences. All disagreements were deliberated and the definitions for each taxonomy category were updated to avoid further misunderstandings. Afterwards, each annotator labeled half of the remaining set independently of each other. Whenever the annotators were unsure about the appropriate category for a sentence they marked the sentence as *unsure* and labeled it with the category they thought would suit best. Afterwards, the other annotator labeled all sentences that were marked as *unsure* by the original annotator. For the cases were the second annotator was also unsure about the category, both annotators discussed the final labeling and a decision was made. In total there were 88 sentences (6.2% of the whole truth set) where at least one annotator was unsure about the labeling, indicating that most of the times the annotators were confident about their work. The disagreement for the unsure cases was of 2.81%.

TABLE IV
PERCENTAGES OF LABELED SENTENCES IN THE TRUTH SET

Category	# Reviews	Proportion
Information Seeking	101	0.07107671
Information Giving	583	0.41027445
Feature Request	218	0.15341309
Problem Discovery	488	0.34342013
Others	31	0.02181562
Total	1421	1

Only 31 sentences were labeled in the *other* category. i.e., 2.18% of the truth set, indicating that our taxonomy covers most of the evolution topics discussed in sentences that are informative for developers. After this annotation process our truth set comprised 1390 sentences.

Table IV shows the number of reviews in the truth set that were labeled as belonging to a certain category. *Information giving* was the most common category, making 41% of the truth-set, *problem discovery* followed with 34% of the truth-set, whereas *feature request* and *information seeking* were only present in 15% and 7% of the sentences respectively. The truth-set is used to generate the training and test sets for the machine learning phase of our approach. Specifically, we used 278 items from our fully manually labeled truth set (20%) as a training set for the different ML techniques we employed, while the remaining 1112 sentences (80%) of the truth set constituted the test set.

2) Used Metrics: We evaluate our results using the precision, recall, and F-measure metrics commonly used in machine learning. In our evaluation we compare the human generated truth set with the automatically generated classification. For each category, the correctly classified items have been computed as *true positives*, the items incorrectly labeled as belonging to that specific category have been considered *false positives* and the items incorrectly labeled as belonging to other categories have been computed as *false negatives*. Precision is computed by dividing the number of *true positives* by the sum of *true positives* and *false positives*. Recall is computed by dividing the number of *true positives* by the sum of *true positives* and *false negatives*. We compute the F-measure by using its general form definition, which returns the harmonic mean of the precision and recall.

3) Statistical Tests: In order to compare if the differences between the different input features and classifiers were statistically significant we performed a Friedman test, followed by a post-hoc Nemenyi test, as recommended by Demšar [14].

IV. RESULTS

A. RQ1: Are the language structure, content and sentiment information able to identify user reviews that could help developers in accomplishing software maintenance and evolution tasks?

Table V gives an overview of the main results obtained through different configuration of machine learning algorithms: (i) NLP features only, (ii) TA features only, (iii) both NLP and SA features, (iv) both NLP and TA features, (v) all NLP, SA, and TA features. For reason of space the table does not report the results achieved when learning the ML techniques by using only SA features since in that case we obtain the worst results with a precision and recall that never exceeds the threshold of 20% and 10% respectively. These results are not surprising because SA features are characterised by only three possibile values, that are insufficient to assign the reviews to one of the four categories of our taxonomy.

The results in Table V show that the NLP+TA+SA configuration had the best results with the J48 algorithm, among all possible feature inputs and classifiers with 75% precision and 74% recall. Therefore, we base the forthcoming result analysis on the NLP+TA+SA configuration with the J48 classifier.

Table VI shows the precision, recall and F-Measure for each category (see Section II-B) obtained through the J48 algorithm, using the NLP+TA+SA features. In particular, *problem discovery* was the class with the highest F-measure, followed by the *information giving* and *information seeking* categories. On the other hand, the *feature request* category was the category with the lowest F-measure. This mirrors the high performance obtained in terms of both precision and recall by three of the categories (average precision of 76% and average recall of 79%). The only exception is the *feature request* category where the precision is 70% but the recall value is very low (23%). This means that for *feature request* the classifier marks relevant sentences accurately, but not all sentences belonging to that category are detected. The results suggest that app users very often rely on common/recurrent patterns, successfully detected by our approach, when their *intention* is to communicate a bug or a problem. On the other hand, app users can request new features in many different ways, making it hard to identify common patterns to detect them. The outcome can also be in part explained by the low amount of *feature requests* in the truth set (see Table IV). Indeed, *information seeking* and *feature request* are the least assigned categories in the truth

TABLE V
RESULTS OF COMBINATION OF NLP, TA AND SA APPROACHES

	NLP			TA			NLP + SA			NLP + TA			NLP + TA + SA		
Classifier	Precision	Recall	F-Measure	Precision	Recall	F-Measure	Precision	Recall	F-Measure	Precision	Recall	F-Measure	Precision	Recall	F-Measure
Bayes	0.572	0.661	0.609	0.665	0.584	0.545	0.572	0.661	0.609	0.687	0.677	0.65	0.691	0.683	0.655
SVM	0.577	0.662	0.61	0.592	0.614	0.584	0.643	0.658	0.639	0.679	0.684	0.666	0.676	0.682	0.664
Logistic Regression	0.577	0.662	0.61	0.462	0.46	0.457	0.561	0.643	0.585	0.492	0.492	0.485	0.453	0.419	0.427
J48	0.577	0.662	0.61	0.572	0.58	0.563	0.726	0.73	0.702	0.696	0.687	0.664	**0.752**	**0.742**	**0.72**
ADTree	0.697	0.67	0.63	0.619	0.611	0.591	0.79	0.719	0.672	0.713	0.707	0.694	0.79	0.719	0.672

set. While the *information seeking* category involves easily recognisable structures (e.g. the question mark at the end of the sentence, or the use of known words as *"how"* or *"what"*), *feature requests* are more complex to detect because of the variety of structures, words and sentiments they could implicate. Thus, the use of a larger set of *feature requests* in the truth set could improve the performance for this kind of sentences. However, observing Table VII, that shows examples of sentences and the related categories that were assigned to them by J48 (using the NLP+TA+SA features), we can notice that such classifier detects with high precision very useful *feature requests* posted by users of mobile apps from a developer perspective.

TABLE VI
RESULTS BY CATEGORY FOR THE J48 ALGORITHM

Category	Precision	Recall	F-Measure
Feature Request	0.704	0.225	0.341
Problem Discovery	0.875	0.776	0.822
Information Seeking	0.712	0.684	0.698
Information Giving	0.68	0.904	0.776
Weighted Avg.	0.752	0.742	0.72

TABLE VII
EXAMPLES OF SENTENCES AND THEIR CATEGORY AS CLASSIFIED BY THE J48 ALGORITHM

Sentence	Category
They just need to update the layout I fill like everything is hidden I want a better task bar.	feature request
Please restore a way to open pin links in external browser or let us save photos.	feature request
App crashes when new power up notice pops up.	problem discovery
Please fix the syncing issues with the iPad app.	problem discovery
It's already possible to rearrange boards why not the pins on a single board?	Information seeking
Overall it is fun and provides a lot of good info.	Information giving
This app runs so smoothly and I rarely have issues with it anymore.	Information giving

B. RQ2: Does the combination of language structure, content and sentiment information produce better results than individual techniques used in isolation?

To answer RQ2 we train ML techniques using different combinations of TA, SA and NLP features as shown in Table V. Among the different kinds of features used individually, the NLP features allow to obtain the best results with the alternating decision tree (ADTree) classifier with 70% precision and 67% recall, although the results of the other machine learning algorithms are also positive (58% precision and 66% recall). On the other hand, the single TA input achieves the best results with the alternating decision tree (ADTree) classifier (62% precision and 61% recall), and the worst results through the Logistic Regression technique with a precision and a recall of 46%. Furthermore, as discussed in the previous section, sentiment information alone is insufficient to classify reviews. Nevertheless, SA in combination with the other techniques adds valuable information that allow classifiers to improve their performances (see Table V).

The NLP+TA configuration achieves the best results through the alternating decision tree (ADTree) classifier with both precision and recall of 71% while the NLP+SA configuration obtains the best results through the J48 classifier with both precision and recall of 73%. For all the considered configurations, Logistic Regression and Naive Bayes proved to be the worst techniques to identify relevant sentences from a software maintenance and evolution perspective, while the ADTree and J48 classifiers are the best. As discussed in the previous section, the best performance is achieved by the NLP+TA+SA combination with the J48 classifier.

These results are encouraging, if we consider that we used just 20% of our dataset to train the different ML algorithms and predicted in the remaining 80%. Indeed, using a larger number of points in the training set (e.g., using 40% of our dataset as a training set and the remaining 60% as a test set) is likely to result in higher performances for *feature requests* and, in general, for all the categories. Table VIII reports the precision, recall and F-measure values achieved when training the J48 classifier using different combinations of features and varying the size of the training set (20%, 40% and 60%). In general, with the exception of some fluctuations (few cases in which we have a slightly decreases of the performances) the performances achieved by the J48 algorithm improve with the addition of more training data. Specifically, when J48 is trained using combinations of features such as, NLP, TA, NLP+SA, NLP+TA the improvements in terms of F-measure range between 2% and 12%. The best results are achieved by the J48 classifier with the combination of NLP+SA features when the size of the training set is 60%: 85% precision, 85% recall and 84% F-measure. This result is interesting because it suggests that for classifying user reviews content with high precision and recall it is sufficient to rely on approaches that extract information about the *sentiment* and *intention (structure)* of a user review. Clearly, such results also confirm the importance of training the classifier with a larger training set. To evaluate the performances of the proposed approach we also randomly sampled 20% of the dataset as training

TABLE VIII
RESULTS BY CATEGORY FOR THE J48 ALGORITHM WHEN VARYING THE SIZE OF THE TRAINING SET.

	NLP			TA			NLP + SA			NLP + TA			NLP + TA + SA		
Training set %	Precision	Recall	F-Measure	Precision	Recall	F-Measure	Precision	Recall	F-Measure	Precision	Recall	F-Measure	Precision	Recall	F-Measure
J48-20	0.577	0.662	0.61	0.572	0.58	0.563	0.726	0.73	0.702	0.696	0.687	0.664	0.752	0.742	0.72
J48-40	0.736	0.683	0.654	0.624	0.638	0.623	0.737	0.725	0.724	0.812	0.802	0.796	0.743	0.721	0.717
j48-60	0.684	0.651	0.639	0.603	0.612	0.591	**0.845**	**0.847**	**0.838**	0.678	0.66	0.657	0.743	0.721	0.717

data and repeated the process 100 times. Thus, we learned the J48 classifier using (i) *NLP+TA* and (ii) *NLP+SA+TA* sets of features respectively. The results of the J48 classifier are again pretty positive and stable: when the J48 classifier is trained using *NLP+TA* our approach achieves, in average, 79% precision, 78% recall and 77% F-Measure; when the J48 classifier is trained using *NLP+SA+TA* our approach achieves, in average, 80% precision, 80% recall and 79% F-Measure. Moreover, we used (i) *NLP+TA* and (ii) *NLP+SA+TA* sets of features to perform a ten-fold cross validation and our results are again quite stable: 80% (or above) precision, 80% (or above) recall and 79% (or above) F-Measure.

The results of the Friedman test revealed that the difference in performance among the classifiers is not statistical significant in terms of F-Measure. Thus, we can conclude that when comparing classifiers' performance and using different input configurations, the choice of the classifier does not affect the results in a significant manner.

To analyse how the language structure, content and sentiment information affected the classification results we executed a Friedman test on the F-Measure scores obtained by the J48 algorithm for each possible input combination. The test concluded that the difference in the results when having different inputs is statistically significant ($p-value = 0.007$). To gain further insight about the groups that are statistically different from each other we performed a Nemenyi test. The test revealed that there is a marginally significant difference between the TA and NLP+TA+SA combinations ($p-value = 0.09$). Moreover, it also highlighted a statistically significant difference between results achieved when relying only on TA features and when using NLP+SA features ($p - value = 0.0061$). This result confirms the importance of NLP and SA features over TA features when classifying reviews into categories relevant to maintenance and evolution tasks.

V. THREATS TO VALIDITY

Threats to construct validity concern the relationship between the theory and the observation. For the truth set creation we rely on error-prone human judgement, because there is a level of subjectivity in deciding if a sentence falls within a specific category. To alleviate this issue we built a truth set based on the judgement of two annotators. Furthermore, definitions for each of the labeling categories were presented and discussed. Moreover, an initial set of 20 sentences was preliminarily labelled by both annotators and all disagreements were discussed between them. Additionally, annotators marked all of the sentences where they were unsure of the assigned category and then the other annotator re-labelled these sentences, increasing the confidence level of the truth set.

Threats to internal validity concern any confounding factor that could influence our results. A threat to internal validity could involve the taxonomy we selected for classifying the sentences, as the categories could present intersections among them. To alleviate this issue we inferred an initial set of categories by observing the communications occuring among developers and then tried to match these categories with topics often treated in app reviews (see Section II-B), as well as with categories from previously defined taxonomies. Moreover, we assumed that each recognised sentence belongs to only one of the categories we defined. Sentences often contain a variety of intents: a major intent and some minor intents (i.e. *How can I access to my profile? Please fix the bug*). Thus, for each sentence we tried to focus on the major intent (i.e. *problem discovery*) and discarded the minor intents (i.e. *information seeking*) to allow developers to understand user needs and opinions more easily and prioritize their work accordingly. However, some information could be lost and this is therefore a threat to construct validity in our work. Another threat to the internal validity can be represented by the problem of the tests overfitting of the machine learning. To handle this problem we (i) randomly sampled 20% of the dataset as training data and repeated the process 100 times and (ii) applied a ten-fold cross validation. Also in this case the proposed approach obtains good results (see Section IV).

Threats to external validity concern the generalisation of our findings. A threat to the external validity could be represented by the particular apps we selected to extract reviews used in our experimentation. Experimental results may be applicable only on the extracted reviews. To reduce this issue we selected seven different apps belonging to six different app categories from two different app stores with different characteristics (see Section III-A).

VI. RELATED WORK

Harman et al. [23] introduced app store mining and analyzed technical and business aspects of apps by extracting app features from the official app descriptions. Chandy and Gu [9] classified spam in the AppStore through a latent model capable of classifying apps, developers, reviews and users into the normal and malicious categories. Pagano and Maalej [31] investigated the types of user feedback present in user reviews useful for developers. We map some of their findings into the labels we used in this work.

Iacob and Harrison [24] extracted feature requests from app reviews by means of linguistic rules and used Latent Dirichlet Allocation (LDA) [6] to group the feature requests. Differently from this work, we employed linguistic rules, text analysis,

and sentiment analysis to mine a wider range of information from user reviews (not only feature requests). LDA was also used for: (i) feature based sentiment analysis of reviews [20], (ii) user reviews summarization [18], and (iii) the identification of incorrectly rated reviews [16]. In contrast with these past works, we also investigated text structure and sentiment dimensions and proved that the analysis of these dimensions could overcome some of the limitations of traditional lexicon-based approaches. Chen et al. [10] used Naive Bayes for finding informative review sentences and LDA for grouping sentences with similar content. They then rank the groups of reviews. In our study we filtered non-informative reviews using Chen's et al. approach and used a combination of techniques to identify relevant sentences. Similarly to Chen et al. we could rank the sentences that are considered more important in each category. Li et al. [28] analyze user reviews to measure user satisfaction by matching words or phrases with a predefined dictionary. Our purpose is to overcome the limitations coming from standard words matching. Khalid et al. [25] mined the Apple iOS App Store, focusing on reviews with one- or two-star ratings, in order to categorize the types of complaints by users and evaluate how complaints affect ratings while Bavota et al. [4] empirically demonstrated the relationship between the success of apps (in terms of user ratings), and the change- and fault-proneness of the underlying APIs. Our work is rather focused in investigating how different techniques could be combined in order to catch in an automated way useful information from app reviews for accomplishing maintenance and evolution tasks.

Bacchelli et al. [2] presented an approach to extract useful information from development emails i.e. text, junk, source code, patch and stack traces. Previous works addressed the problem of bugs misclassification in issue trackers [1], [26] building ML classifiers which relying on textual features in bug reports try to classify (or reclassify) the issues. Several works focused on the API documentation trying to: (i) categorize source code and textual descriptions in API discussion forums [38], (ii) detect *knowledge items* in API reference documentation [11], or (iii) infer formal method specifications from API documents [32]. Sharama et al. [35] proposed a new approach based on a language model that can help developers in identifying software related tweets. Panichella et al. [34], [36] mined bug reports, development mailing lists and StackOverflow in order to discover descriptions that can suitably explain methods.

VII. Conclusions and Future Work

In this paper we presented an approach which uses Natural Language Processing, Sentiment Analysis and Text Analysis in order to detect and classify sentences in app user reviews that could guide and help app developers in accomplishing software maintenance and evolution tasks. The classification is performed according to a taxonomy of sentences categories deduced by analysing reviews and development emails. Results of our study show that the combination of NLP, TA and SA techniques allows to detect useful sentences for app developers with appreciable levels of precision (75.2%) and recall (74.2%). We also proved that some configurations substantially improve both precision and recall when increasing the size of the training set. Additionally, we found that a classifier trained with *structure* (with NLP) and *sentiment* (with SA) features performs significantly better than when only trained with *text* (with TA) features.

Results also highlighted that *structure* (with NLP), *sentiment* (with SA), and *text* (with TA) features contained in user reviews could be useful in extracting not only sentences which mention specific topics, but in understanding the *intentions* of the writers concerning the mentioned topics. This, in our opinion, could allow developers to (i) filter relevant information in user reviews, (ii) understand more quickly the software maintenance tasks to apply, and, consequently, (iii) be more responsive to users requests. As a first direction for future work we plan to extend our study to a larger number and variety of apps, involving several developers of different communities to empirically confirm the benefits that developers can achieve when relying on automated *intention mining*. Furthermore, we also plan to complement our approach with topic modeling techniques. Specifically, topics models can be used to cluster sentences in each of the categories of our taxonomy (Section II-B). For example, grouping together sentences of the category *feature request* that are related to the same functionality of a given app. Finally, we also plan to improve our approach by adding more NLP rules and by experimenting with others ways to combine NLP, SA and TA techniques.

Acknowledgments

We thank Francesco Mercaldo (Ph.D. Student at the University of Sannio) for his help in the results' manual validation. Sebastiano Panichella gratefully acknowledges the Swiss National Science foundation's support for the project *"Essentials"* (SNF Project No. 200020−153129).

References

[1] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, Y. Guhneuc, *Is it a bug or an enhancement?: a text-based approach to classify change requests*. CASCON, 2008:23.

[2] A. Bacchelli, T. Dal Sasso, M. D'Ambros, and M. Lanza. *Content classification of development emails*. In Proceedings of the 34th International Conference on Software Engineering (ICSE), 2012, pp. 375-385.

[3] V. R. Basili, L. C. Briand, and W. L. Melo, *A validation of object oriented design metrics as quality indicators,* IEEE Trans. Software Eng., vol. 22, no. 10, pp. 751-761, 1996.

[4] G. Bavota, M. L. Vásquez, C. E. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk, *The Impact of API Change- and Fault-Proneness on the User Ratings of Android Apps*. IEEE Trans. Software Eng. 41(4), pp.384-407, 2015.

[5] M. Bezerra, A. L. I. Oliveira, and S. R. L. Meira, , *A constructive rbf neural network for estimating the probability of defects in software modules,* in Neural Networks, 2007. IJCNN 2007. International Joint Conference on, 2007, pp. 2869-2874.

[6] D. M. Blei, A.Y. Ng, and M. I. Jordan, *Latent dirichlet allocation*, in Journal of Machine Learning Research (JMLR), Vol. 3, 2003, pp. 993-1022.

[7] D. Cer, M.C. de Marneffe, D. Jurafsky, and C.D. Manning, *Parsing to Stanford dependencies: Trade-offs between speed and accuracy*, in Proceedings of the 7th International Conference on Language Resources and Evolution (LREC), 2010.

[8] E. Ceylan, F. Kutlubay, and A. Bener, *Software defect identification using machine learning techniques,* in Software Engineering and Advanced Applications (SEAA), 2006, pp. 240-247.

[9] R. Chandy and H. Gu. *Identifying spam in the iOS app store*. In Proceedings of the 2nd Joint WICOW/AIRWeb Workshop on Web Quality (WebQuality), 2012, pages 56-59.

[10] N. Chen, J. Lin, S.C.H. Hoi, X. Xiao, B. Zhang, *AR-miner: mining informative reviews for developers from mobile app marketplace*. In Proceedings of the 36th International Conference on Software Engineering (ICSE), 2014, pp. 767-778.

[11] Y. B. Chhetri and M. P. Robillard, *Recommending Reference API Documentation*, in Empirical Software Engineering, 2014. To appear

[12] I. Dagan, O. Glickman, and B. Magnini, *The PASCAL recognizing textual entailment challenge*, in Proceedings of The First International Conference on Machine Learning Challenges: evaluating Predictive Uncertainly Visual Object Classification, and Recognizing Textual Entailment, 2005, pp. 177-190.

[13] M.C. de Marneffe, B. MacCartney, and C.D. Manning, , *Generating typed dependency parses from phrase structure parses*, in Proceedings of LREC, 2006, pp. 449-454.

[14] J. Demšar, *Statistical comparisons of classifiers over multiple data sets*, in Journal of Machine Learning Research v.7, 2006, pp. 1-30.

[15] W. B. Frakes and R. Baeza-Yates, *Information Retrieval: Data Structures and Algorithms?*. Prentice-Hall, Englewood Cliffs, NJ, 1992.

[16] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh. *Why people hate your app: Making sense of user feedback in a mobile app store.*. In Proc. of the International Conference on Knowledge Discovery and Data Mining (KDD), 2013, pages 1276-1284.

[17] K. Fundel, R. Küffner, and R. Zimmer, *RelEx - Relation extraction using dependency parse trees*, in Bioinformatics, v.23, n.3, 2007, pp. 365-371.

[18] L. V. Galvis Carreno and K. Winbladh. *Analysis of user comments: an approach for software requirements evolution*. In Proceedings of the 2013 International Conference on Software Engineering (ICSE), 2013, pages 582-591.

[19] B. Glaser, and A. Strauss. *The discovery of grounded theory: Strategies of qualitative research*. New York, NY:Aldine de Gruyter, 1967.

[20] E. Guzman, and W. Maalej *How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews*. In Proceedings of the 22nd IEEE International Requirements Engineering Conference (RE), 2014, pp. 153-162

[21] A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. van Deursen, *Communication in open source software development mailing lists*, in Proceedings of the 10th Working Conference on Mining Software Repositories (MSR), 2013, pp 277-286.

[22] A. Guzzi, A. Begel, J.K. Miller, and K. Nareddy, *Facilitating Enterprise Software Developer Communication with CARES*, in Proceedings of the 34th International Conference on Software Engineering (ICSE), 2012, pp 1367-1370.

[23] M. Harman, Y. Jia, and Y. Zhang. *App store mining and analysis: MSR for app stores*. In Proc. of the Working Conference on Mining Software Repositories (MSR) 2012, pages 108-111.

[24] C. Iacob and R. Harrison. *Retrieving and analyzing mobile apps feature requests from online reviews*. In Proc. of the Working Conference on Mining Software Repositories (MSR), 2013, pages 41-44.

[25] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan. *What Do Mobile App Users Complain About?*. IEEE Software, vol.32, no. 3, 2015, pp. 70-77.

[26] P. S. Kochhar, F. Thung, and D. Lo. *Automatic Fine-Grained Issue Report Reclassification*. in Proceedings of the 19th International Conference on Engineering of Complex Computer Systems (ICECCS), 2014, pp. 126-135.

[27] O. Kucuktunc, B. B. Cambazoglu, I. Weber, and H. Ferhatosmanoglu. *A large-scale sentiment analysis for Yahoo! Answers*, In Proceedings of the International Conference on Web Search and Data Mining (WSDM), 2012, pp 633-642.

[28] H. Li, L. Zhang, L. Zhang, and J. Shen. *A user satisfaction analysis approach for software evolution*. In Proc. of the Progress in Informatics and Computing Conference (PIC), 2010, volume 2, pages 1093-1097.

[29] Y. Liu, T. M. Khoshgoftaar, and N. Seliya, , *Evolutionary optimization of software quality modeling with multiple repositories,* IEEE Trans. Softw. Eng., vol. 36, no. 6, Nov. 2010, pp. 852-864.

[30] J. Nivre, L. Rimell, R. McDonald, and C. Gómez-Rodríguez, *Evaluation of dependency parsers on unbounded dependencies*, in Proceedings of COLING, 2010, pp. 813-821.

[31] D. Pagano, and W. Maalej *User Feedback in the AppStore: An Empirical Study*. In Proceedings of the 21st IEEE International Requirements Engineering Conference (RE), 2013, pp.125-134.

[32] R. Pandita, X.Xiao, H. Zhong, and T. Xie, *Inferring method specifications from natural language API descriptions*, in Proceedings of the 34th International Conference on Software Engineering (ICSE), 2012, pp. 815-825

[33] B. Pang, L. Lee, and S. Vaithyanathan, *Thumbs up?: sentiment classification using machine learning techniques*, Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing, pp. 79-86, 2002.

[34] S. Panichella, J. Aponte, M. Di Penta, A. Marcus, and G. Canfora *Mining source code descriptions from developers communications*, in Proceedings of the 20th IEEE International Conference on Program Comprehension, 2012, pp. 63-72.

[35] A. Sharma, Y. Tian, D. Lo. *NIRMAL: Automatic identification of software relevant tweets leveraging language model*. In Proceedings of the 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, (SANER), 2015, pp. 449-458.

[36] C. Vassallo, S. Panichella, M. Di Penta, and G. Canfora, *CODES: mining source code description from developers discussions*, in Proceedings of the 22th IEEE International Conference on Program Comprehension (ICPC), 2014, pp. 106-109.

[37] I. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.

[38] Y. Zhou, Y. Tong, R. Gu, H. Gall, *Combining Text Mining and Data Mining for Bug Report Classification?*. In Proceeding of 30th International Conference on Software Maintenance and Evolution (ICSME), 2014, pp. 311-320.

[39] T. Zimmermann and N. Nagappan, , *Predicting defects with program dependencies*, in Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on, 2009, pp. 435-438.